

SEMINÁRIO PYTHON

Bruno Campos
William Müller

HISTÓRICO

Início da
Implementação

Python 0.9.0
alt.sources

Python 2to3

Python 3.6.1

1989

1991

2008

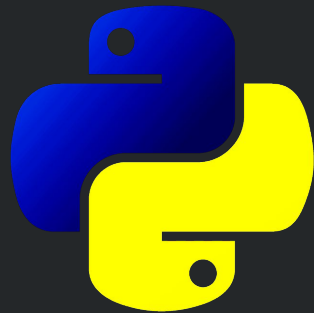
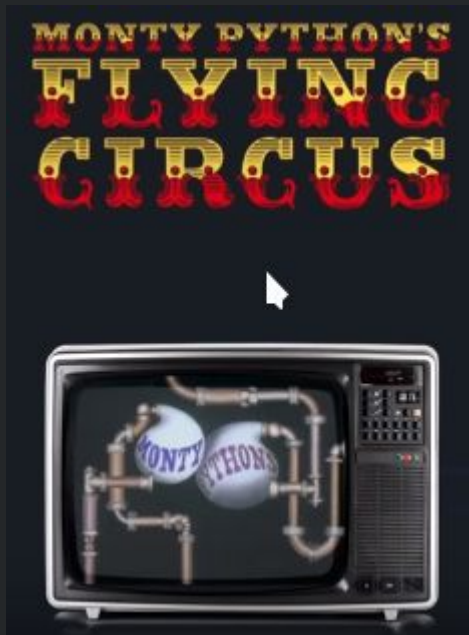
2017

Guido van Rossum

reestruturação

baseada na
linguagem ABC

Nome e Fundação

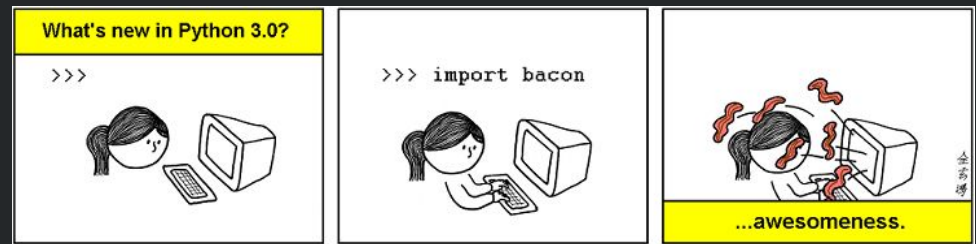


python
SOFTWARE FOUNDATION

O que é o Python

Características

- Linguagem de propósito geral
- Fácil e intuitiva
- Multiplataforma
- Livre
- Organizada
- Multiparadigma
- Tipagem dinâmica e forte



Prioriza o programador

O intuito de Rossum era criar uma linguagem **interpretada** com comandos simples e fáceis de entender, pois segundo ele, programas em C são grandes e complicados de entender para novos programadores.

Características

Portabilidade

Padronização
Multi-plataforma

Generalidade

Vários Domínios Mobile, Web,
Desktop Data Mining, Jogos

Compilador de bytecode

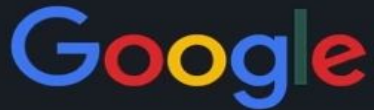
- Também pode compilar através de sua VM gerando um bytecode(.pyc ou .pyo)
- Utilizando o interpretador interativo não é necessário a criação do arquivo de Python compilado

Zen of Python

>>> import this

- Bonito é melhor que feio.
- Explícito é melhor que implícito.
- Simples é melhor que complexo.
- Complexo é melhor que complicado.
- Linear é melhor do que aninhado.
- Esparso é melhor que denso.
- Legibilidade conta.
- Casos especiais não são especiais o bastante para quebrar as regras.
- Ainda que praticidade vença a pureza.
- Erros nunca devem passar silenciosamente.
- A menos que sejam explicitamente silenciados.
- Diante da ambiguidade, recuse a tentação de adivinhar.
- Deveria haver um – e preferencialmente só um – modo óbvio para fazer algo.
- Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
- Agora é melhor que nunca.
- Embora nunca frequentemente seja melhor que *já*.
- Se a implementação é difícil de explicar, é uma má idéia.
- Se a implementação é fácil de explicar, pode ser uma boa idéia.
- Namespaces são uma grande idéia – vamos ter mais dessas!

Quem usa Python?



Quem usa Python?



PROJETO

Paradigmas

Procedural
Funcional
Orientado a Objetos

Domínios

A linguagem suporta numeros muito grandes

- Científico
- Biotecnologia
- Inteligência Artificial

PARADIGMAS

Procedural

Utiliza comandos de atribuição e segue a ordem do código durante a execução

```
1 def cubo(x):  
2     res = x**3  
3     return res  
4  
5 print cubo(20)
```

Funcional

Utiliza funções aplicadas a determinados parâmetros como principal meio de execução

```
1 vetor = [2, 18, 9, 22, 17, 24, 8, 12]  
2 print filter(lambda x: x ** 3, vetor)
```

PARADIGMAS

Orientação a Objetos

É tudo objeto

Há suporte para polimorfismo, e herança (inclusive herança múltipla).

```
1 class Aluno:
2     def __init__(self, nome):
3         self.nome = nome
4
5 eu = Aluno("Allisson")
6 print eu.nome
```

Em Python não existe encapsulamento. Convenciona-se que atributos com o nome começando com um `_` são de uso privado da classe

Exemplos

Científico

Astropy
Biopython
Numpy
TomoPy

Comercial

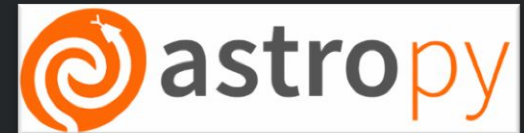
GNU Mailman
Django
Kivy

Inteligência Artificial

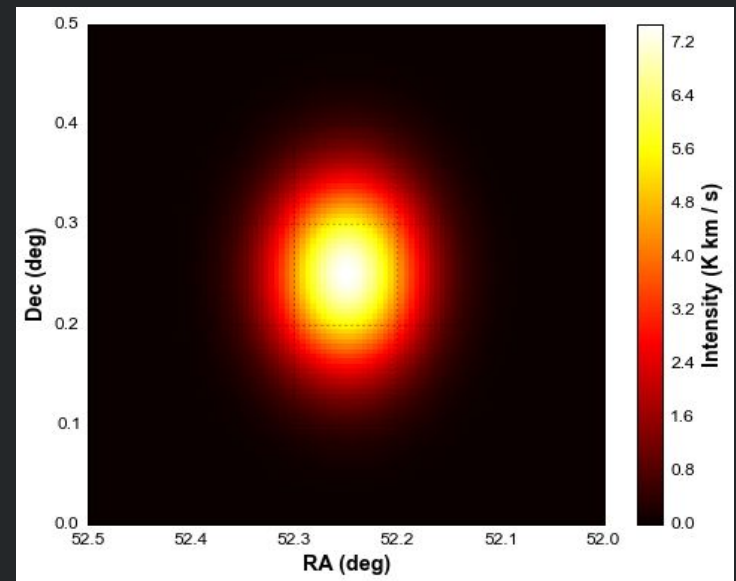
Scikit-learn

DOMÍNIOS

Científico



```
>>> from numpy import *  
>>> abs(-1)  
1  
>>> abs(array([-1.2, 1.2]))  
array([ 1.2, 1.2])  
>>> abs(1.2+1j)  
1.5620499351813308
```

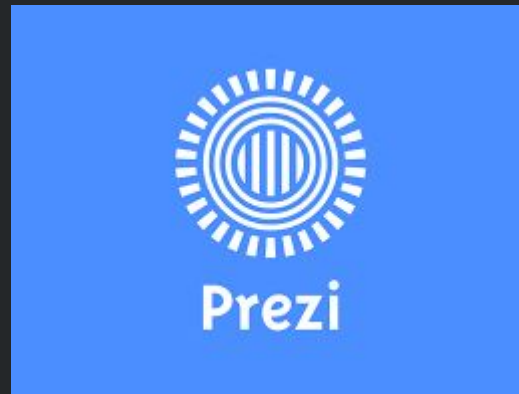


DOMÍNIOS

Comercial

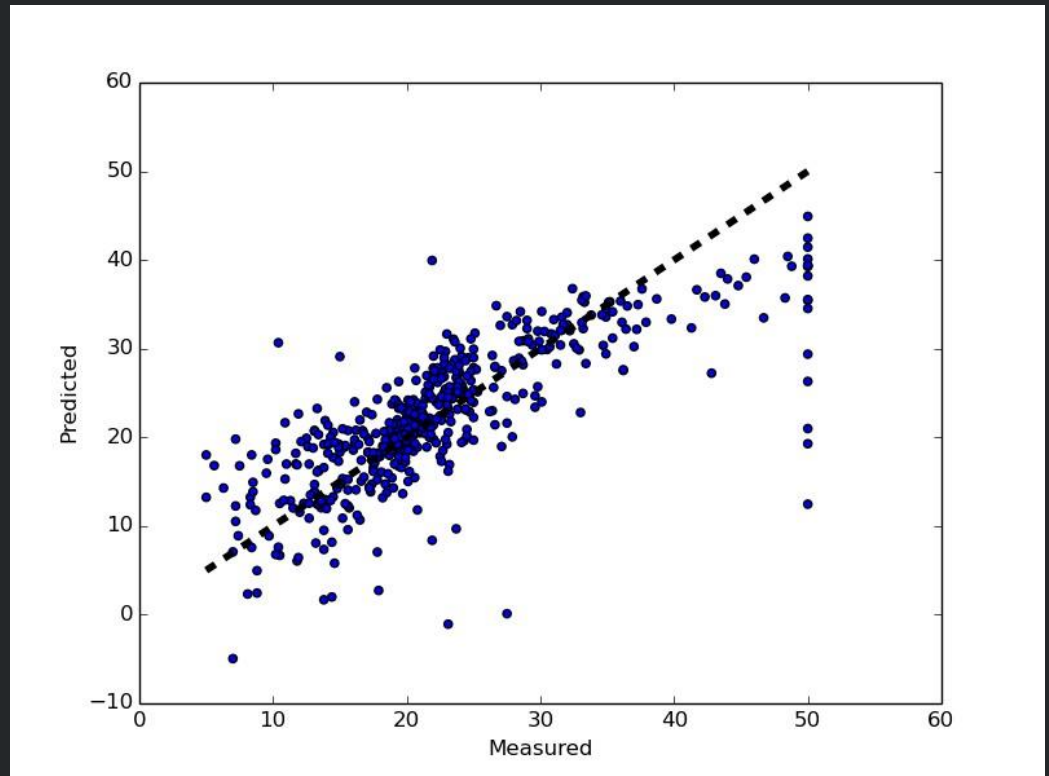


Conjunto de ferramentas em Python para agilizar o desenvolvimento web.



DOMÍNIOS

Inteligência Artificial



ESTRUTURAS

Tipos built-ins

Boolean

Int / Float / Complex

Char / String

List

Dictionary

Comandos

if / else for while

try / except

Funções

def

Classes

Funções Anônimas

Outros

Palavras

reservadas

Variáveis

Operadores

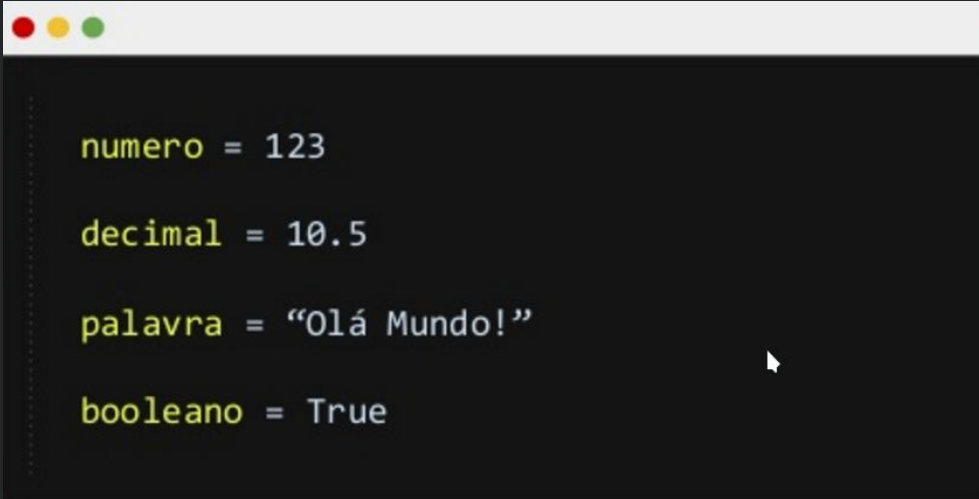
Indentação

ESTRUTURAS

Variáveis

Não precisa declarar o tipo

É o conteúdo da variável que determina



```
numero = 123  
decimal = 10.5  
palavra = "Olá Mundo!"  
booleano = True
```

ESTRUTURAS

Tipos built-ins

- int - para números inteiros
- str - para conjunto de caracteres
- bool - armazena True ou False
- list - para agrupar um conjunto de elementos
- tupla - igual ao tipo list, porém, imutável
- dic - para agrupar elementos que serão recuperados por uma

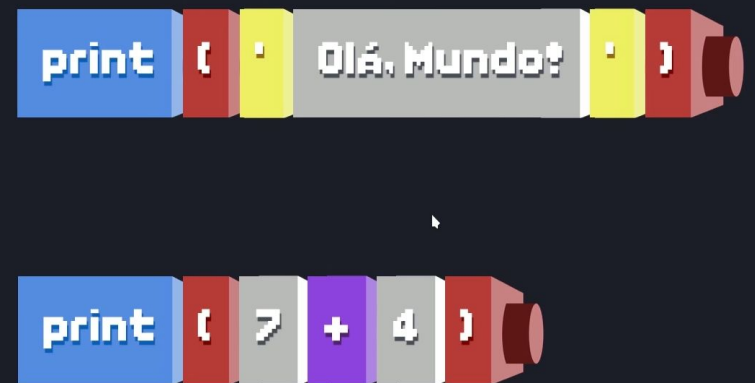
```
1 numero = 12.3
2 complexo = complex(3, 4)
3 booleano = True
4 string = "My String"
5 lista = ['item 1', 'item 2']
6 tupla = ('python', 'java', 1, 2.5)
7 dicionario = {'chave': 123}
```

ESTRUTURAS

Funções

def

```
1 def adicionar(x, y):  
2     return x + y  
3  
4 a = 5  
5 b = 3  
6 print adicionar(a, b)  
7 # 8
```



The image shows two Scratch-style code blocks. The top block is a 'print' block (blue) with a 'text' input field (yellow) containing the text 'Olá, Mundo!'. The bottom block is a 'print' block (blue) with a 'math' input field (purple) containing the expression '7 + 4'.

ESTRUTURAS

Funções

Classes

```
1 class Carro(object):
2     def __init__(self, chassi):
3         self.chassi = chassi
4
5     def imprimir_tipo(self):
6         print "Sou um carro."
7
8 class Ferrari(Carro):
9     def imprimir_marca(self):
10        print "Sou uma Ferrari."
11
12 ferrari = Ferrari('123456')
13 ferrari.imprimir_tipo()
14 # Sou um carro.
15 ferrari.imprimir_marca()
16 # Sou uma Ferrari.
```

ESTRUTURAS

Comandos

If / elif / else

while

for

```
1 resposta = 42
2 if resposta > 42:
3     print "Maior que 42."
4 elif resposta < 42:
5     print "Menor que 42."
6 else:
7     print "Igual a 42."
8 # Igual a 42.
```

```
1 contador = 0
2 while contador < 5:
3     print contador,
4     contador += 1
5     # 0 1 2 3 4
6
7 for i in range(5):
8     print i,
9     # 0 1 2 3 4
```

ESTRUTURAS

Comandos

try except

```
1 x = 10
2 try:
3     num = x / 0
4 except ZeroDivisionError:
5     print ("Nao dividirás por 0!")
```

ESTRUTURAS

Outros

Palavras
reservadas

| | | | | |
|--------|--------|--------|----------|---------|
| and | del | from | not | while |
| as | elif | global | or | with |
| assert | else | if | pass | yield |
| break | except | import | print | class |
| exc | in | raise | continue | finally |
| is | return | def | for | lambda |
| try | False | True | nonlocal | |

ESTRUTURAS

Outros

Operadores (lógicos, relacionais e aritméticos)

```
1 a, b = True, False
2 print a and b
3 # False
4 print a or b
5 # True
6 print not a
7 # False
```

```
1 a, b = 1, 2
2 print a == b
3 # False
4 print a != b
5 # True
6 print a > b
7 # False
8 print a < b
9 # True
10 print a >= b
11 # False
12 print a <= b
13 # True
```

```
1 a, b = 4, 2
2 print a + b
3 # 6
4 print a - b
5 # 2
6 print a * b
7 # 8
8 print a / b
9 # 2
10 print a % b
11 # 0
12 print a ** b
13 # 16
```


ESTRUTURAS

Indentação

- Sem delimitadores de bloco
- Para manter a fácil leitura, não é usado pontuação como em outras linguagens.
- Indentação é obrigatória

```
1 resposta = True
2 if resposta:
3     print "Resposta"
4     print "True"
5 else:
6     print "Resposta"
7     print "False"
8 # Resposta
9 # True
```

LEGIBILIDADE

Simplicidade Global

Poucos componentes básicos

Poucas palavras reservadas

Bom

| | | | | |
|--------|--------|--------|----------|---------|
| and | del | from | not | while |
| as | elif | global | or | with |
| assert | else | if | pass | yield |
| break | except | import | print | class |
| exc | in | raise | continue | finally |
| is | return | def | for | lambda |
| try | False | True | nonlocal | |

LEGIBILIDADE

Tipos de Dados

Bom

Facilidade para criar tipos

```
1  class Teste:  
2      def foo(bar):  
3          #algum codigo  
4      end  
5  
6      def foo2(bar):  
7          #algum codigo  
8      end  
9      #falta de informações sobre o  
10     #retorno e o tipo dos parâmetros  
11     #dificulta a legibilidade
```

LEGIBILIDADE

Tipos de Dados

Ruim

Tipagem dinâmica

```
1 #cria uma lista
2 a = [66.25, 333, 333, 1, 1234.5]
3 #cria um dicionario
4 tel = {'jack': 4098, 'sape': 4139}
5 #cria uma tupla
6 t = 12345, 54321, 'hello!'
```

LEGIBILIDADE

Aspectos da Sintaxe

Bom

Indentação por blocos

```
1 x = True
2   if x:
3       print ("Verdadeiro!")
4   else:
5       print ("Falso!")
```

```
1 def cubo(x):
2     res = x**3
3     return res
4
5 print cubo(20)
```

CAPACIDADE DE ESCRITA

Abstração

Bom

Funcional

Orientada a Objetos

```
1 lista = [x**2 for x in range(10)]
```

```
1 class Classe:  
2     def foo(bar):  
3         print ("Olá")  
4     end
```

CAPACIDADE DE ESCRITA

Expressividade

Bom

Funcional

Orientada a Objetos

```
1 class Conta:
2     saldo = 0
3
4     def __init__(self, saldo):
5         self.saldo = saldo
6     def __add__(self, other):
7         return Conta(self.saldo + other.saldo)
```

```
1 from conta import Conta
2
3 conta = Conta(10)
4 conta2 = Conta(20)
5 conta3 = conta + conta2
6 print (conta3.saldo)
```

CUSTO

Execução

Alto

- Python
- Java
- Ruby

